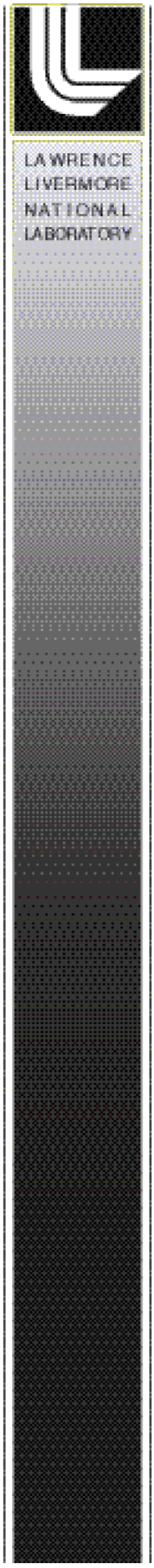


LLNL-TR-402283

Solving a million equations

Derek E. Decker

March 14, 2008

Disclaimer

Auspices Statement

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

There is a method that's been around for over a half dozen years which involves capturing a time sequence of images using masked silicon imaging arrays that can shift rows very quickly. For an example of this, see the 2001 paper entitled "MHz Class Repetitively Q-Switched, High-Power Ruby Lasers for High-Speed Photographic Applications" by Peter E. Nebolsine, D. R. Snyder, and J. M. Grace. Before describing this method, it may help to understand a similar technology.

Frame-transfer and interline transfer CCDs use the masked regions to store an image. These regions can be used to store images by rapidly shifting photoelectrons (electrons generated by photons interacting in silicon) created in the unmasked regions to a location in the silicon under the masked regions through a bucket brigade method common to Charge Coupled Devices (CCDs). In frame-transfer and inter-line transfer cameras, about 50% of the image sensor is masked off. These CCDs are sometimes referred to having electronic shutters. Two frames can be acquired if other shutters or pulsed illumination is employed. Clearly, the interline method is faster because only one row of shift is required to store the image behind a masked region. See Figure 1 below.

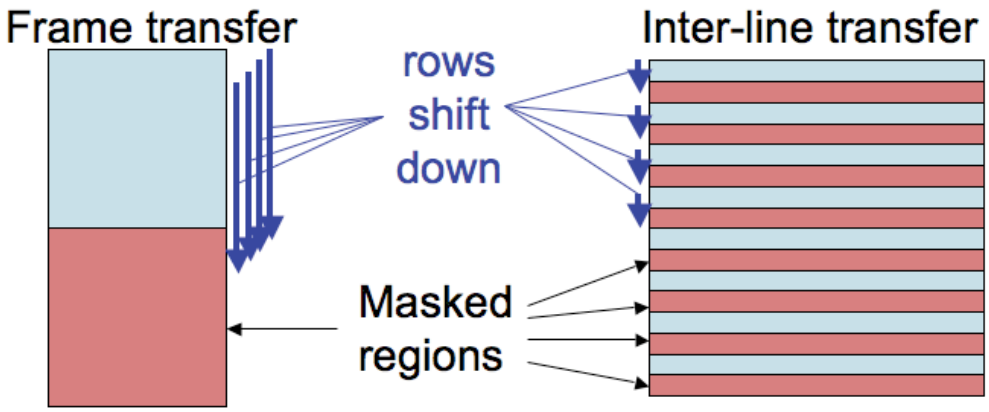


Figure 1

Given a fixed number of available pixels one can put on a silicon chip, you can trade off spatial resolution for additional frames. Two types of masks are shown in Figure 2 below for Multi-Line Transfer (MLT). In Figure 2, one-fifth (or 20%) of the CCD is exposed while the rest (80%) is masked off. Depending on external shuttering or lighting conditions, four or five frames can be captured in rapid succession with these masks as long as your camera can shift down four rows quickly during the experiment. Unless the chip is specially designed with rectangular pixels, the "Distorted MLT" mask maintains horizontal resolution and gives up vertical resolution which results in an image that can appear distorted if not properly displayed (pixels shown would not be square or the image would appear compressed). One can obtain the same spatial resolution in both axes by using an array of apertures as shown in the "Tilted MLT" of Figure 2.

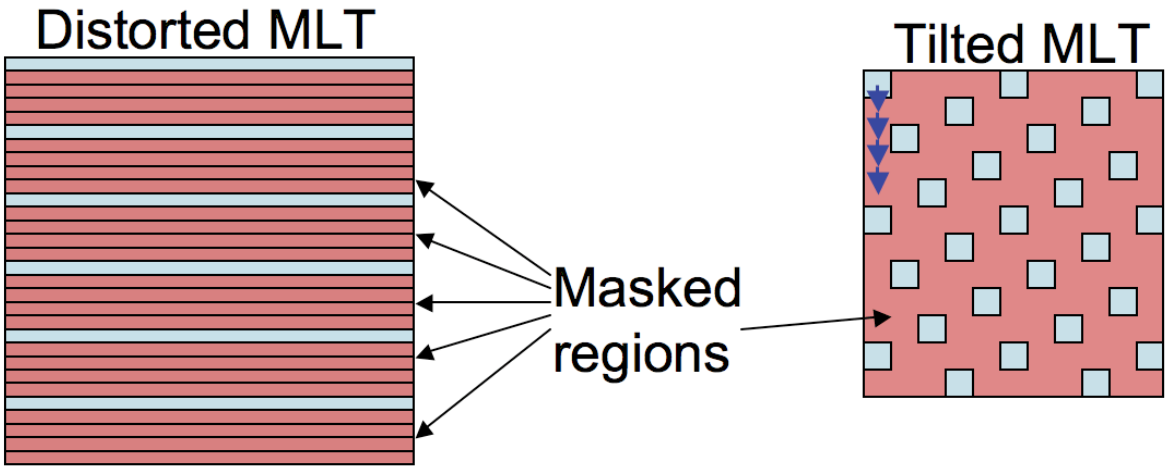


Figure 2

If you assume the photoelectrons are shifted down, then the first image will be at the bottom of each column and the most recent (or last) image is at the top (where the aperture is). Software is subsequently used to un-shuffle the image data. See Figure 3 below.

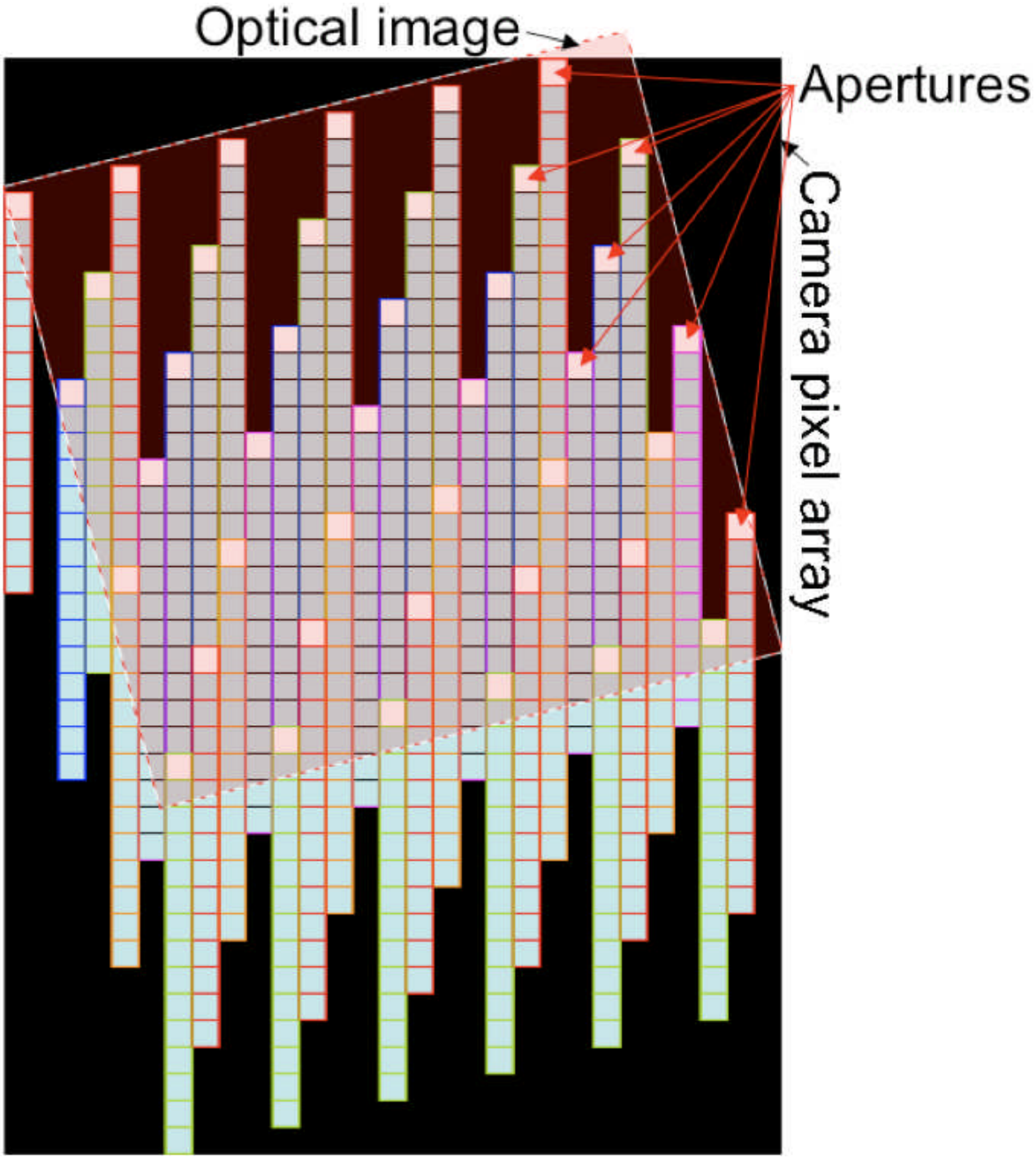


Figure 3

The top row of apertures in figure 3 shows six columns color coded in red, one column for each aperture. After transferring photoelectrons down 14 rows, it is essentially full and must stop and wait for cessation of light (such as the closing of another shutter). One can think of this method of creating a time streak record of each image pixel (defined by an aperture). If light continued to enter this camera after 14 row shifts, then the top row (in red) would over-write the fourth row (in orange). And, the second row (in green) would over-write the fifth row (in red). And, the third row (in blue) would over-write the sixth row (in green). In a real system, you'd want hundreds of rows (such as might be available in a megapixel imaging chip) and the over-writing problem affects nearly all of your pixels.

Unshuffling the data is a relatively easy task. However, if light leaks from an aperture, meant to create photoelectrons in one pixel only, into adjacent pixels around it, data corruption occurs. Fortunately, one can model this corruption with the help of a calibration image and solving numerous equations. In our particular example, we use the Salvador Imaging camera employ the Dalsa FT50 megapixel chip which can shift rows at 5 MHz. Thus, we can acquire images at a rate of 5 million frames per second {Mfps}.

Figure 4 shows a portion of an image produced by imaging a chrome on glass mask onto a maskless chip. We have also used the more optically efficient lenslet arrays to create the same pattern of spots on an unmasked CCD chip. These methods, with imperfect imaging, leads to spots blurring into blobs and light leaking beyond the intended pixel for each illumination spot.

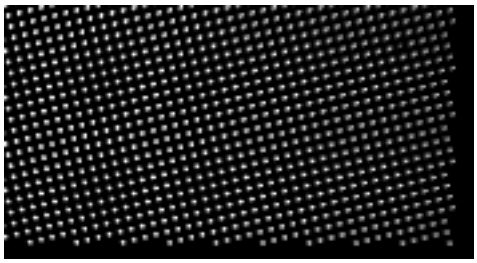
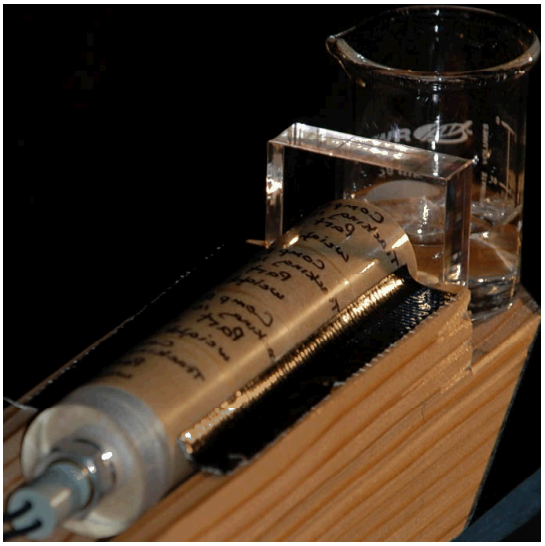


Figure 4



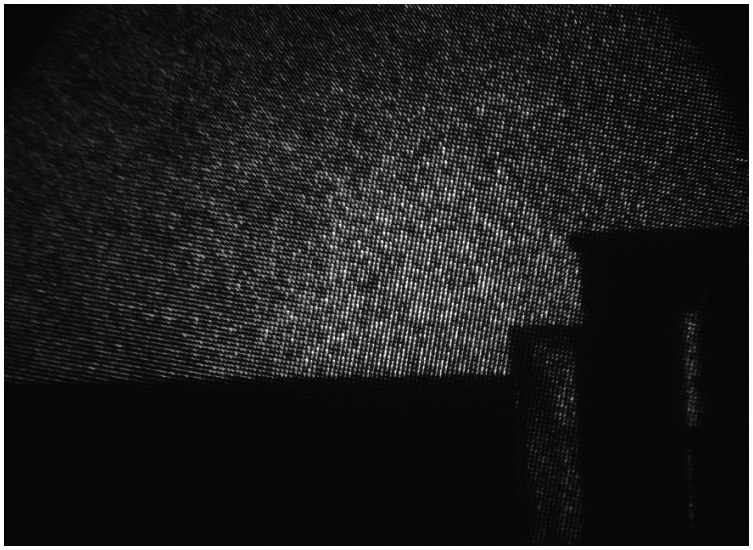


Figure 5

Figure 5 above shows a cylinder of high explosives (HE). An exploding bridgewire on the left will begin the detonation and a cone of expanding smoke will emerge from the cylinder of HE as time progresses. Eventually, the shock will enter Acrylic plate and beaker half filled with water.

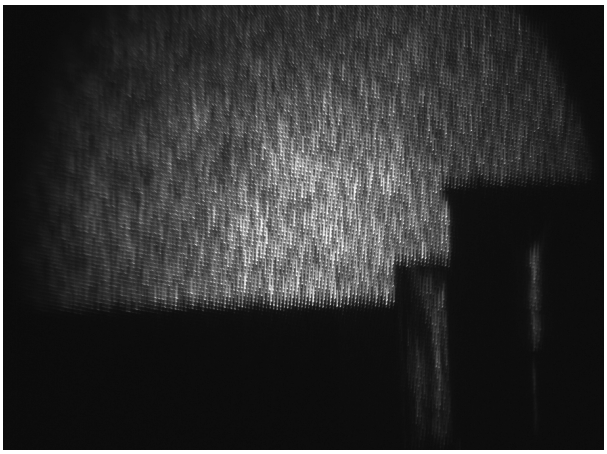


Figure 6a

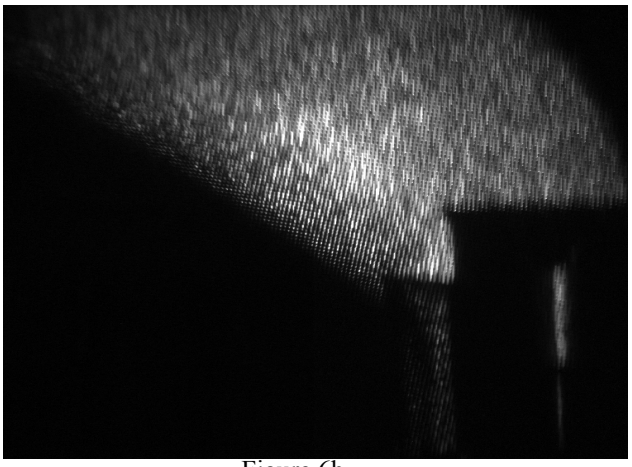


Figure 6b

Figure 6a shows a pre-shot image in which the spots (or blobs) are streaked (22 rows shifted up, not down). The bright spots at the bottom of each streak is the result of continued illumination by a pulsed laser. The camera continuously cleans out (dumps or drains away) the photoelectrons prior to the command to begin recording. However, once recording starts and shifting is completed, light continues to generate excess photoelectrons in the pixels below the apertures (or at the illumination spots). Figure 6b shows the obscuration of light back illuminating the HE as the cone of smoke moves left to right. Notice that the light columns to the right are still 22 pixels high but the height is decreased as you approach the edge of the moving smoke, as you would expect because light was blocked at the beginning of the streak recording. The bright spots at the bottom of each column were minimized by using another HE charge to break a fiber optic carrying the illuminating laser light. This shuttered the light more quickly after streaking had ceased.

There are really two coding challenges. The first involves automatically segmenting the pre-shot, calibration image of spots (see Figure 5 and Figure 7) into numbered blobs and properly assigning every pixel to a blob. Experience suggests we can assume the blobs do not overlap. The 12-bit values from each pixel identified as belonging to a particular blob define that blob's surface topography. Those ratios of pixel values to the blob peak value are assumed to remain constant throughout the experiment. Those ratios define constants in the equations.

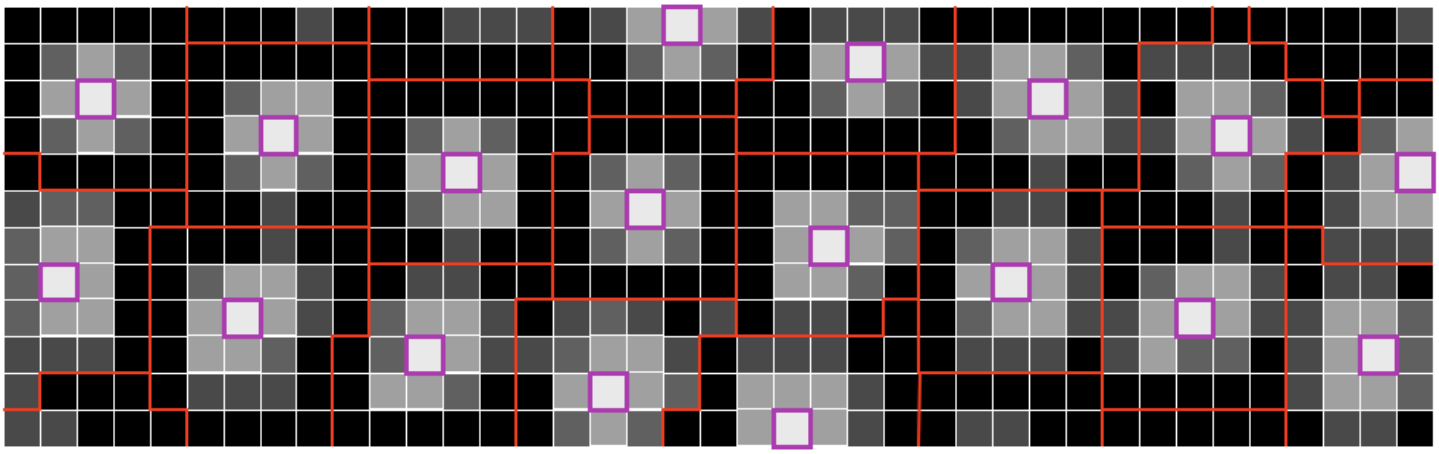


Figure 7

The second coding challenge involves constructing and solving the million equations, one for each pixel of the megapixel chip. Each pixel value in the shot image (see Figure 6b), is the sum of 22 components (due to the vertical shifting of 22 rows). Going back to the idea that rows of photoelectrons are being shifted down, you can see in Figure 3 that the illuminated pixels at the bottom (first image) of the bottom row of blobs are uncorrupted. Likewise the top pixels (last image) of the top row of blobs (blue columns of Fig. 8) are also uncorrupted by other blobs (yellow and red columns of Fig. 8).

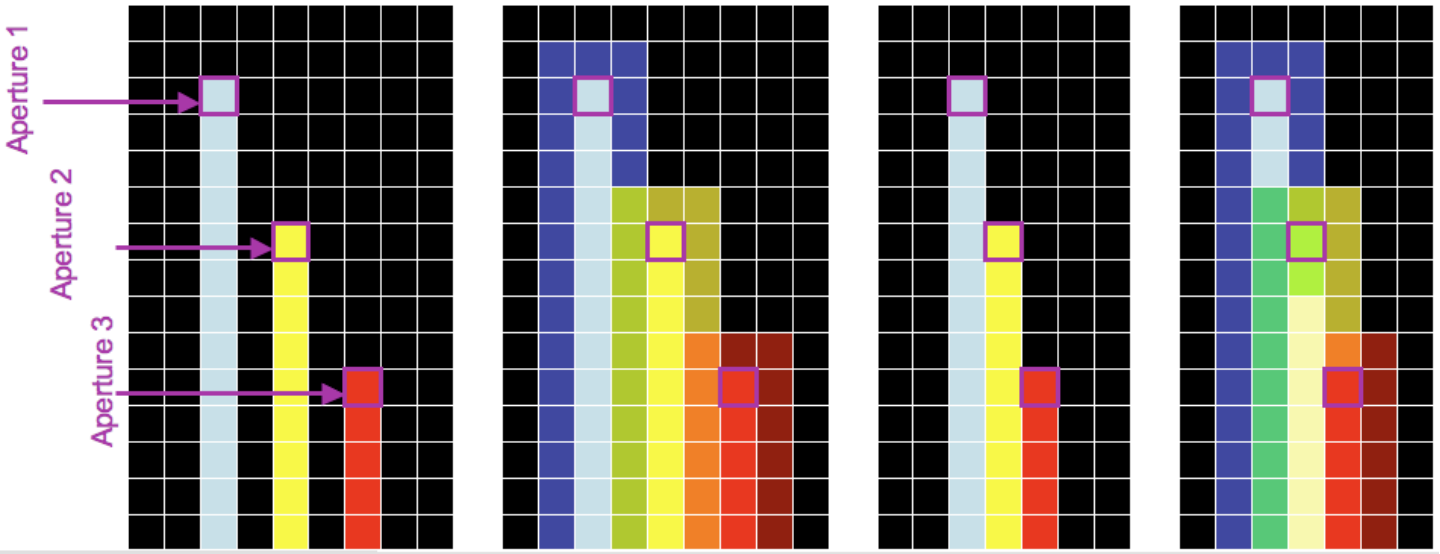


Figure 8a Figure 8b Figure 8c Figure 8d

Figures 8a and 8c have no leakage. Figures 8b and 8d show leakage which is due to imperfect imaging of the apertures to the CCD pixels. One can extract uncorrupted data easily from 8a and 8b but half the recording space is wasted resulting in half the spatial resolution obtained by sacrificing every other column. In reality, 8d is what we have to decifer. To complicate matters, the intensity changes along the vertical column which stores the time history of each aperture. In each of the four figures, three apertures are illustrated (with a purple perimeter). These are only three of the "image pixels" and the total number of "image pixels" comprise a small fraction (about 213 x 213) of the 1024 x 1024 "CCD pixels".

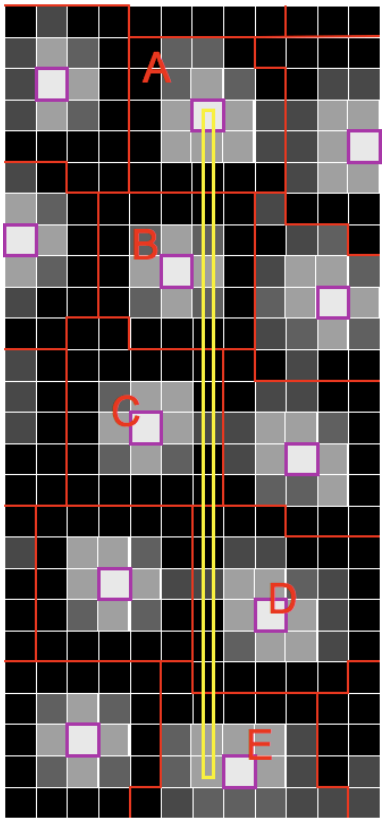


Figure 9

A yellow rectangle in Figure 9 will help illustrate how one of the equations could be formed. The first photoelectrons that contribute to that "image pixel" (the aperture of region A) for the first frame, is $I_A(x_A,y_A,t_1)$. At time t_2 the photoelectrons are moved down a row and there is an additional contribution from that same aperture (I_A) but as a lower off-peak intensity due to blur. That ratio is known from a pre-shot calibration image. The second element of the equation is thus $I_A(x_A,y_{A-1},t_2)$. The third element of the sum is $I_A(x_A,y_{A-2},t_3)$. Now we enter a new region (B) whose peak intensity $I_B(x_B,y_B)$ is reduced by the known ratio $I_B(x_{B+1},y_{B+2})/I_B(x_B,y_B)$.

The shot image looks like rain (Figure 6b) due to the streaking method. Using the pre-shot image of Figure 9, the pixel intensity at the bottom of the yellow rectangle in Figure 9 is the sum of 22 components, namely …

$$\begin{aligned} I(x_{E-1},y_E) = \quad & I_A(x_A,y_A,t_1) + I_A(x_A,y_{A-1},t_2) + I_A(x_A,y_{A-2},t_3) + \\ & I_B(x_{B+1},y_{B+2},t_4) + I_B(x_{B+1},y_{B+1},t_5) + I_B(x_{B+1},y_B,t_6) + I_B(x_{B+1},y_{B-1},t_7) + I_B(x_{B+1},y_{B-2},t_8) + \\ & I_C(x_{C+1},y_{C+2},t_9) + I_C(x_{C+1},y_{C+1},t_{10}) + I_C(x_{C+1},y_C,t_{11}) + I_C(x_{C+1},y_{C-1},t_{12}) + I_C(x_{C+1},y_{C-2},t_{13}) + \\ & I_D(x_{D+1},y_{D+3},t_{14}) + I_D(x_{D+1},y_{D+2},t_{15}) + I_D(x_{D+1},y_{D+1},t_{16}) + I_D(x_{D+1},y_D,t_{17}) + I_D(x_{D+1},y_{D-1},t_{18}) + I_D(x_{D+1},y_{D-2},t_{19}) + \\ & I_E(x_{E+1},y_{E+2},t_{20}) + I_E(x_{E+1},y_{E+1},t_{21}) + I_E(x_{E+1},y_E,t_{22}) \end{aligned}$$

This could be done for all of the 1024 x 1024 pixels of the "shot image". The notation I'm using may be misleading because the coordinate for x_{E+1} is actually $x_E + 1$ (one pixel to the right of the blob E peak x-coordinate x_E).

Notice, that the pixel just above the last one we calculated is …

$$\begin{aligned} I(x_{E-1},y_{E+1}) = \quad & I_A(x_A,y_{A+1},t_1) + I_A(x_A,y_A,t_2) + I_A(x_A,y_{A-1},t_3) + I_A(x_A,y_{A-2},t_4) + \\ & I_B(x_{B+1},y_{B+2},t_5) + I_B(x_{B+1},y_{B+1},t_6) + I_B(x_{B+1},y_B,t_7) + I_B(x_{B+1},y_{B-1},t_8) + I_B(x_{B+1},y_{B-2},t_9) + \\ & I_C(x_{C+1},y_{C+2},t_{10}) + I_C(x_{C+1},y_{C+1},t_{11}) + I_C(x_{C+1},y_C,t_{12}) + I_C(x_{C+1},y_{C-1},t_{13}) + I_C(x_{C+1},y_{C-2},t_{14}) + \\ & I_D(x_{D+1},y_{D+3},t_{15}) + I_D(x_{D+1},y_{D+2},t_{16}) + I_D(x_{D+1},y_{D+1},t_{17}) + I_D(x_{D+1},y_D,t_{18}) + I_D(x_{D+1},y_{D-1},t_{19}) + I_D(x_{D+1},y_{D-2},t_{20}) + \\ & I_E(x_{E+1},y_{E+2},t_{21}) + I_E(x_{E+1},y_{E+1},t_{22}) \end{aligned}$$

As long as we have the same time t_i and the same blob j, then we can reduce the number of variable by inserting ratios from the precalibration image. For example, at time t_1, the first element of each of the two equations above is related by a known ratio, namely, $I_A(x_A,y_A)/ I_A(x_A,y_{A+1})$ which comes from the topography of blob A. This greatly reduces the number of variables that need to be solved for. Such an array of equations may form an over determined sparse matrix. I have access to Mathematica running on a cluster. It's not Grid Mathematica so only one Mathematica core would be running at a time. But, there is 32 GB of RAM available which should make this go faster than my desktop. I am open to other software and/or hardware approaches to solving this problem. Any assistance you are willing to provide will be much appreciated.